**UNIVERZITET U ZENICI**

**POLITEHNIČKI FAKULTET**

**Softversko inženjerstvo**

**Osnove baza podataka**

**ONLINE ISPITNI SISTEM**

**Profesor**: Doc.dr.sc. Denis Čeke **Student:** Ernes Mušija

Zenica, akademska 2022/2023

**SADRŽAJ:**

[UVOD 3](#_Toc124451954)

[Prikupljanje i određivanje korisničkih zahtjeva 4](#_Toc124451955)

[Razvoj baze podataka 6](#_Toc124451956)

[ER Dijagram 6](#_Toc124451957)

[Relacijski model 7](#_Toc124451958)

[Normalizacija 9](#_Toc124451959)

[Prva Normalna Forma (1NF) 9](#_Toc124451960)

[Druga Normalna Forma (2NF) 9](#_Toc124451961)

[Treća normalna forma (3NF) 10](#_Toc124451962)

[Generisanje baze podataka 12](#_Toc124451963)

[Razvoj aplikacije 15](#_Toc124451964)

[Korisnički interface 15](#_Toc124451965)

[Backend 16](#_Toc124451966)

[Zaključak 18](#_Toc124451967)

[Literatura 19](#_Toc124451968)

# 

# UVOD

Cilj ovog projektnog zadatka je razvoj, normalizacija te implementacija baze podataka za web aplikaciju koja omogućava održavanje ispita i kvizova. Pored toga razvijeni su i odgovarajući backend i frontend sistemi čiji je primarni cilj unos podataka u bazu kao i njihova obrada i prikazivanje krajnjim korisnicima.

Pošto je održavanje online ispita osnovni cilj aplikacije, osnovni koncept oko kojeg je čitav sistem baziran jeste potreba da se zadovolje dvije vrste korisnika aplikacije: Zadavači i Polagači ispita. Iz ovog razloga je i čitava struktura aplikacije podijeljena na dva dijela, jedan prilagođen za korisnike čiji su računi označeni kao „Zadavač“ i jedan za račune sa ulogom „Polagač“.

Pored toga osnovni entitet u bazi podataka a i cijeloj aplikacije je sam ispit od kojeg čitav sistem i zavisi.

U nastavku će detaljnije biti opisana baza podataka u samom konceptualnom smislu, zatim će biti objašnjen način razvoja baze što podrazumijeva ER Dijagram, relacijski model, proces normalizacije. Na kraju će biti ukratko objašnjen sam proces razvoja aplikacije u smislu korištenih backend i frontend tehnologija i bilo kojih drugih specifičnosti.

# Prikupljanje i određivanje korisničkih zahtjeva

Konceptualna šema baze podataka:

Korisnici aplikacije mogu praviti, zadavati i polagati ispite. Podaci koje je potrebno sačuvati o korisniku uključuju:

Ime, prezime, datum rođenja, username i šifru. Također čuvamo podatke o autorizacijama koje korisnik posjeduje da bi mogli razlikovati admin korisnike od običnih.

Korisnici koji nemaju admin autorizacije mogu se podijeliti u dvije grupe: Zadavači i polagači ispita.

Zadavači imaju mogućnost:

* Formiranja ispita (gdje je potrebno sačuvati datum formiranja), zadavač može formirati koliko god želi ispita a ispit može imati samo jednog zadavača,
* Bodovanje zadataka i pitanja koji nemaju automatski odgovor (zaokruživanje ponuđenih odgovora ili odgovor sa default tačnim rješenjem), zadatke iz istog ispita može ocijeniti samo njihov zadavač
* Ocjenjivanje ispita, u ovom procesu imamo ternarnu vezu gdje možemo imati jednog zadavača koji ocjenjuje više ispita sa više polagača.
* Dodavanja pitanja u ispit

Podaci specifični za zadavača su: Broj napravljenih ispita i broj neocijenjenih ispita.

Polagači imaju mogućnost polaganja ispita gdje više polagača može polagati više raznih ispita. Njihovi ispiti koje su polagali mogu biti ocijenjeni od strane zadavača tog istog ispita.

Podaci koji se čuvaju za polagače su: Broj položenih ispita, broj polaganih ispita, prosječna ocjena.

Za sam proces polaganja ispita osim poznavanja zadavača i polagača istog, potrebno je sačuvati podatke i o datumu, vremenu početka i vremenu završetka ispita.

Za ocjenjivanje se čuva ocjena, broj osvojenih bodova, datum objavljenog rezultata i moguće napomene koje zadavač ima da napiše polagaču, te podatak o tome da li je ispit neocijenjen.

Ispit je osnovni entitet oko kojeg se vrti čitava baza podataka i ima podatke o: broju pitanja, ocjenama, vremenu trajanja, oblasti iz koje je ispit kao i unikatni ispitni kod od 9 cjelobrojnih karaktera koji se koristi za pristup ispitu.

Ispit se sastoji od jednog ili više pitanja o kojima čuvamo: tekst pitanja, maksimalni broj bodova koje nosi pitanje, moguće vrijeme trajanja koje korisnik ima da odgovori pitanje.

Pitanje se mora vezati za jedan i samo jedan ispit.

Pitanja mogu biti ona sa ponuđenim odgovorom o kojima čuvamo maksimalno 3 ponuđena ne tačna odgovora i jedan tačan odgovor, kao i podatak o izgubljenim bodovima

u slučaju biranja ne tačnog odgovora. Ova pitanja zadavač ne mora ocijeniti, jer ako je odgovor na njih tačan, onda korisnik dobiva maksimalan broj bodova, u suprotnom gubi bodove.

O zadacima čuvamo mogući default tačni odgovor koji automatski daje maksimalne bodove polagaču. Pošto je potrebno ocijeniti ove zadatke, odgovor na njih se čuva u posebnoj tabeli u kojoj je zadatak strani ključ, zajedno sa polagačem i ispitom u kojem se zadatak nalazi.

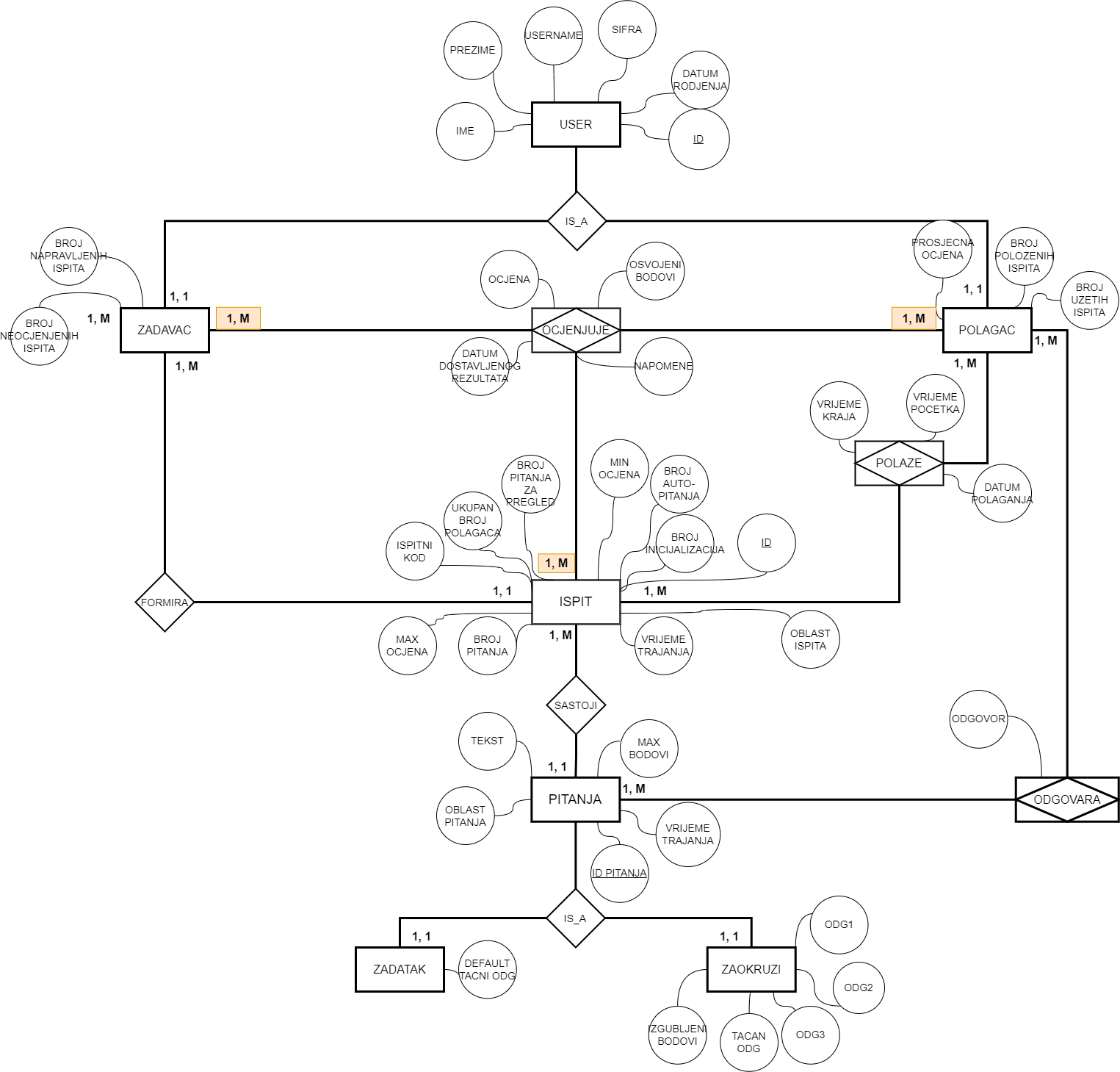
U sljedećem poglavlju prikazan je razvoj ove baze podataka.

# Razvoj baze podataka

U ovom poglavlju je opisan i prikazan ER dijagram, relacijski model, proces normalizacije te krajnji izgled baze podataka prije početka samog generisanja baze podataka sa SQL upitima.

## ER Dijagram

ER dijagram za ovu bazu je prikazan na sljedećoj slici:



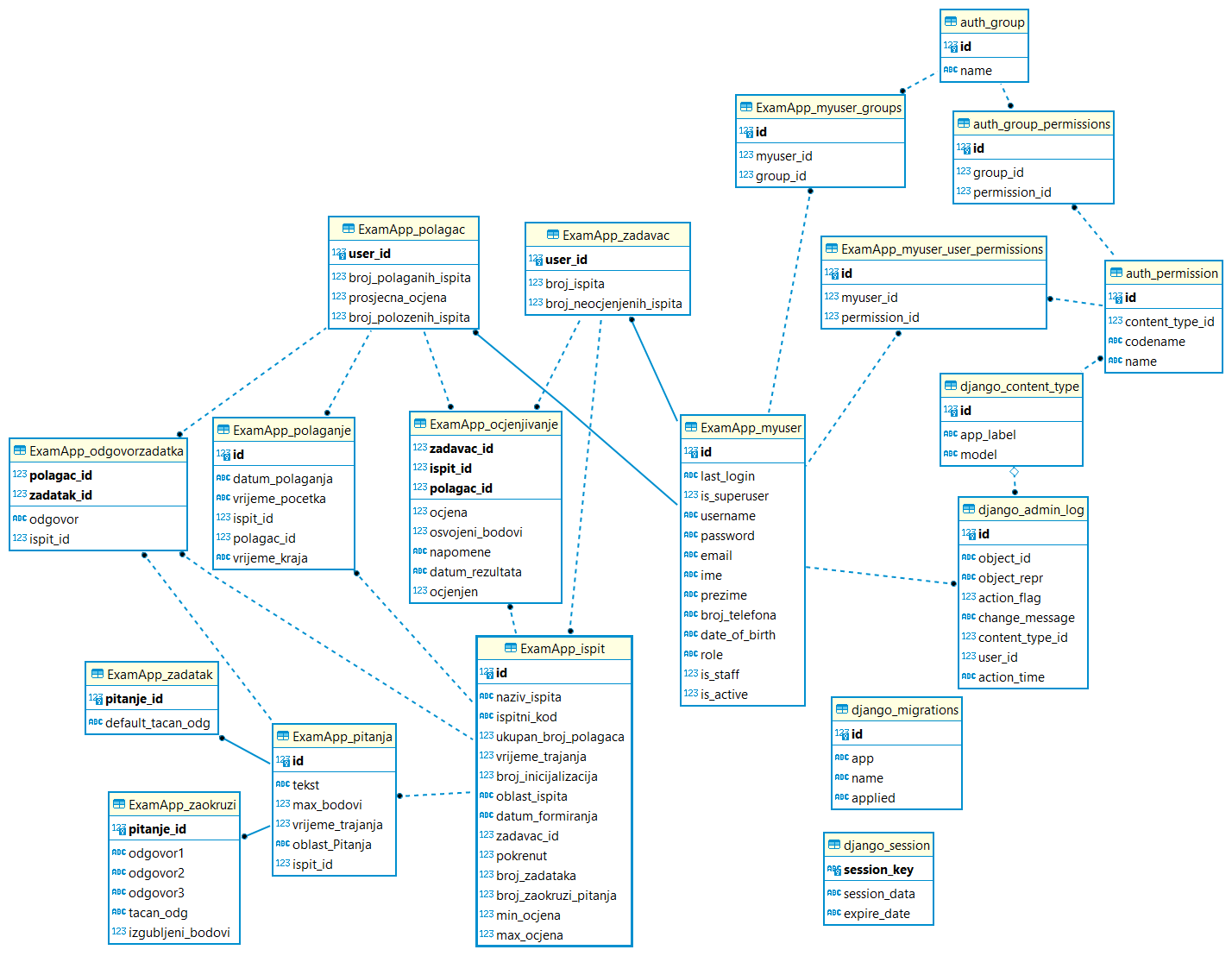
SLIKA 1 – ER DIJAGRAM

Sa dijagrama vidimo da se korisnički model dijeli na Zadavača i Polagača, i ta dva modela su povezana sa ispitom. Takav pristup je bio neophodan jer u aplikacijskom smislu, korisnikovo iskustvo će biti potpuno drugačije zavisno od toga na koji korisnički model mu je račun prijavljen. Između polagača, zadavača i ispita ima ternarna veza za ocjenjivanje. Kardinalitet kod ove veze je urađen u „look aside“ notaciji. Ostatak dijagrama je „look across“.

Entitet pitanja se također dijeli na dva entiteta. Entitet zadatak se razlikuje od zaokruži jer za njega zadavač mora posebno izvršiti bodovanje i ocjenjivanje, dok kod zaokruži možemo automatski izvršiti bodovanje. Iz ovog razloga imamo potrebu za asocijativnim entitetom „odgovara“ u kojem čuvamo polagačev odgovor na zadatak, da bi ga zadavač mogao kasnije bodovati u ocjenjivanju.

## Relacijski model

Na sljedećoj slici je prikazan relacijski model ovog ER dijagrama. Baza je napravljena koristeći Django ORM i SQLite RDBMS. Model je generisan u aplikaciji Dbeaver.



SLIKA 2 – RELACIJSKI MODEL

Na relacijskom modelu, pored entiteta definisanih u ER dijagramu, imamo i nekoliko tabela koje Django ORM automatski generiše. Zbog bolje preglednosti na slici su te tabele su pomjerene u desnu stranu. Te tabele uglavnom čuvaju podatke koji su bitni za autorizaciju, dozvole i uloge koje različiti korisnici imaju. Zbog toga, kao što se i vidi sa slike, te tabele su povezane sa MyUser tabelom, i to im je jedina veza sa ostatkom baze. Imamo i tabelu koja čuva podatke o migracijama koje su učinjene tokom formiranja same baze podataka.

Asocijativni entiteti Ocjenjivanje i OdgovorZadatka imaju kompozitne primarne ključeve koji imaju „unique\_together“ constraint. Kod entiteta polaganje, imamo posebno polje „id“ umjesto kompozitnog ključa, jer postoji mogućnost da jedan korisnik može imati potrebu da više puta polaže isti ispit. To se moglo zaobići dodavanjem „datum\_polaganja“ i „vrijeme\_polaganja“ polja u kompozitni ključ, ali to bi onda dodatno otežalo pisanje upita, a možda i njihovu brzinu izvršavanja. Dodavanjem novog „id“ polja nemamo neke značajne negativne posljedice i dobivamo dosta na jednostavnosti baze.

# Normalizacija

Proces normalizacije baze podataka će biti prikazan u tri poglavlja, svako poglavlje će predstavljati neku od prve tri normalne forme. Automatski generisane tabele neće biti normalizovane (niti ima potrebe da se normalizuju). Koristit će se metoda vertikalne normalizacije.

## Prva Normalna Forma (1NF)

Sve tabele se već nalaze u prvoj normalnoj formi iz sljedećih razloga:

* Svi atribute sadrže samo nedjeljive (atomic) vrijednosti
* Nema atributa koji sadrže skup podataka
* Sve kolone u jednoj tabeli su jedinstvene (nema ponavljajućih grupa kolona)
* Svi neključni atributi funkcijski zavise o primarnom ključu

## Druga Normalna Forma (2NF)

Uslovi za drugu normalnu formu su ti da se tabela već nalazi u 1NF i da postoji potpuna funkcijska zavisnost svih neključnih atributa o primarnom ključu.

Pošto je već dokazano da je cijela baza u 1NF, sada je samo potrebno zadovoljiti drugi uslov.

U praksi, ako nemamo kompozitnih primarnih ključeva, onda je u većini slučajeva svaki neključni atribut automatski potpuno zavisan od primarnog ključa (ovdje je to zasigurno slučaj jer je baza razvijena prateći jasno definisani ER dijagram). Taj uslov je zadovoljen za sljedeće tabele:

* MyUser
* Zadavač
* Polagač
* Ispit
* Pitanja
* Zaokruži
* Zadatak
* Polaganje

Ako bi pojedinačno provjerili ove tabele, vidjeli bismo da su stvarno svi atributi u potpunosti zavisni od svog primarnog ključa.

U tom slučaju, ostaju nam samo tabele OdgovorZadatka i Ocjenjivanje za koje je proces malo detaljnije opisan.

Tabela OdgovorZadatka ima kompozitni ključ od dva atributa: polagac\_id i zadatak\_id. Oba atributa su strani ključevi.

Pored njih, u tabeli imamo ispit\_id, koji je također strani ključ, i funkcionalno je zavisan i od polagača ispita kao i zadatka na koji polagač odgovara.

Atribut odgovor je također potpuno funkcijski zavisan od oba dijela kompozitnog ključa jer moramo znati koji polagač daje odgovor, i na koji zadatak.

Tabela Ocjenjivanje ima tri atributa koji čine kompozitni primarni ključ. Svaki od neključnih atributa je potpuno funkcijski zavisan o svakom dijelu kompozitnog ključa, kao i u prošlom primjeru, tako da i ova tabela ispunjava uslove druge normalne forme.

Ovim zaključujemo da je čitava baza također u 2NF.

## Treća normalna forma (3NF)

Da bismo doveli tabelu u 3NF potrebno je da ona bude u 2NF i da nema tranzitivne zavisnosti neključnih atributa.

Ako postoji tranzitivna zavisnost, onda se uklanja tako što uklanjamo tranzitivno zavisne atribute i stavljamo ih u novu tabelu koju povezujemo sa našom tabelom.

Kao i za prethodne dvije forme, i 3NF je već ispunjena. Provjeru možemo uraditi na tabelama koje su povezane sa ispitom.

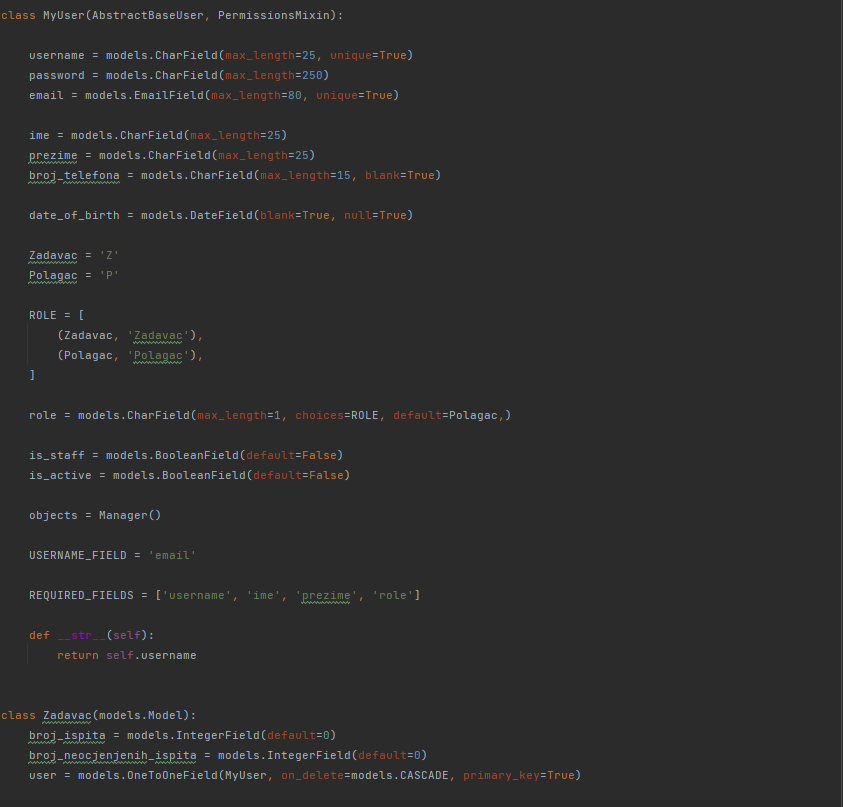
Tabela ocjenjivanje je u 3NF jer su svi atributi funkcijski zavisni od kompozitnog primarnog ključa (ispit\_id, polagac\_id, zadavac\_id) i ni jedan tih atributa nije zavisan od nekog drugo neključnog atributa.

Isto ovo možemo provjeriti i za tabelu polaganje, u kojoj imamo strane ključeve ispit\_id i polagac\_id i primarni ključ „id“. Ne postoji funkcijska zavisnost između ova dva atributa, kao ni zavisnost ostalih atributa (datum, vrijeme kraja i početka) o nekom od ova dva, ali svi su atributi i dalje zavisni od primarnog ključa.

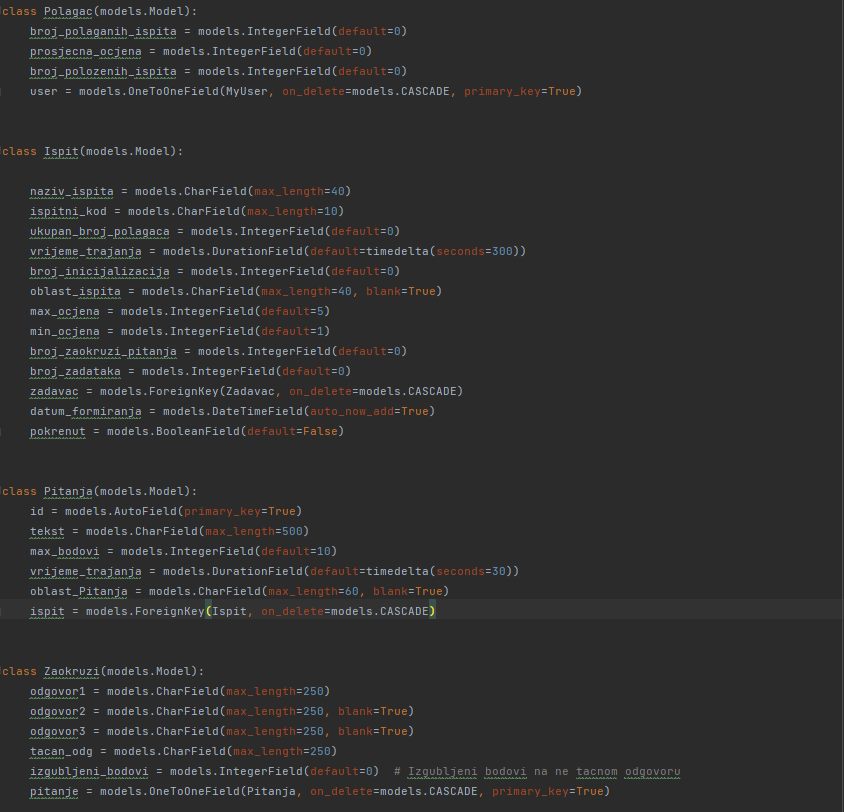
Na isti način kao i kod ova dva primjera možemo opisati i sve ostale tabele u bazi, i tako zaključiti da se nigdje ne pojavljuje tranzitivna zavisnost, pa tako dokazujemo da je cijela baza u trećoj normalnoj formi.

# Generisanje baze podataka

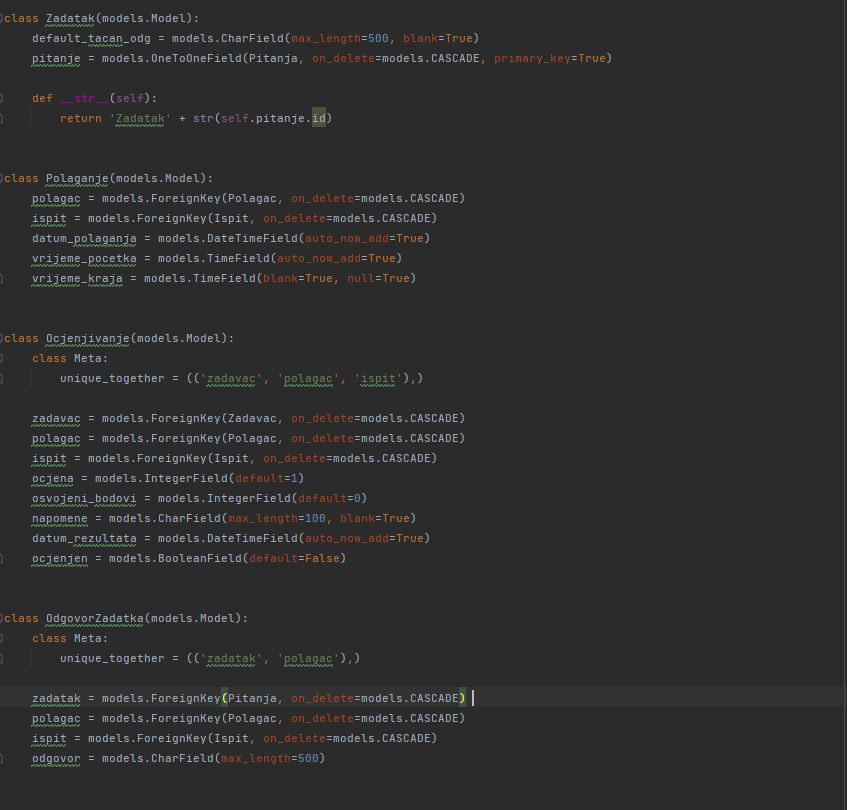
Kao što je već spomenuto, za generisanje baze podataka se koristio Django ORM i SQLite RBDSM. Na sljedećim slikama je prikazan cijeli models.py file u kojem se pišu modeli koji se onda automatski pretvaraju u SQLite upite koji se koriste za generisanje baze.



SLIKA 3 – BAZA KOD 1



SLIKA 4 – BAZA KOD 2



SLIKA 5 – BAZA KOD 3

Model MyUser je nešto drugačije napisan od ostalih modela jer Django ima svoj default korisnički model. Problem sa tim modelom je to što nije optimalan za sve slučajeve, i sama dokumentacija preporučuje da se na početku projekta napravi svoj custom user model koji naslijeđuje od klase defaultnog modela.

Za bilo koja ograničenja (constraints) potrebno je deklarisati class Meta i tu definisati šta želimo. U ovom projektu je jedino korišten unique\_together constraint koji osigurava da kombinacija više atributa mora biti jedinstvenu u svakom objektu koji se kreira za neki entitet.

Null=True osigurava da neki atribut smije imati NULL vrijednost, dok blank=True osigurava da CharField atribut (tekstualni podaci), pri kreiranju nekog objekta, smije ostati prazan.

Kod DateTimeField polja koja imaju auto\_add\_now definisana, pri spremanju objekta, vrijeme u tom polju će biti trenutno vrijeme, prateći vremenskoj zoni korisnika koji je napravio objekat.

Relacije one-to-one i many-to-one se definišu jednostavnim OneToOneField ili ManyToOneField poljima. Postoji i ManyToManyField koji automatski u pozadini kreira pomoćnu tabelu, ali ovdje zbog preglednosti, za many-to-many relacije je ručno kreirana dodatna tabela.

# Razvoj aplikacije

Neke od prednosti korištenja Django frameworka za ovakvu vrstu projekta:

* Fokus projekta je naravno baza podataka, a Django ORM je jedna od najvećih prednosti samo frameworka generalno
* Pošto je ovo „školski“ projekat, Python programski jezik je dobra opcija zbog čitljivosti njegove sintakse
* Django framework je definisan kao backend tehnologija, ali ima dosta alata koji omogućavaju dobro povezivanje sa frontendom, pa možemo ga ček nazvati i Full stack frameworkom u ovakvim situacijama

Za frontend su korišteni HTML i CSS, uz nekoliko Javascript skripti. Nije bilo potrebe za bilo kakvim frontend frameworkom, zbog prethodno opisanih osobina Djanga, kao i cijele poente projekta (razvoj baze podataka).

## Korisnički interface

Sada će biti ukratko opisana aplikacija sa strane korisnika i interfacea koji će biti njemu prezentiran.

Prije nego što se korisnik prijavi na račun, ima samo opciju prijave i registracije, i može pročitati neki osnovni opis aplikacije na početnoj stranici.

Prilikom registracije, pored osnovnih podataka koje korisnik treba dostaviti (ime, prezime, datum rođenja...), također je potrebno odabrati koju vrstu računa želi da napravi (Zadavač ili Polagač ispita).

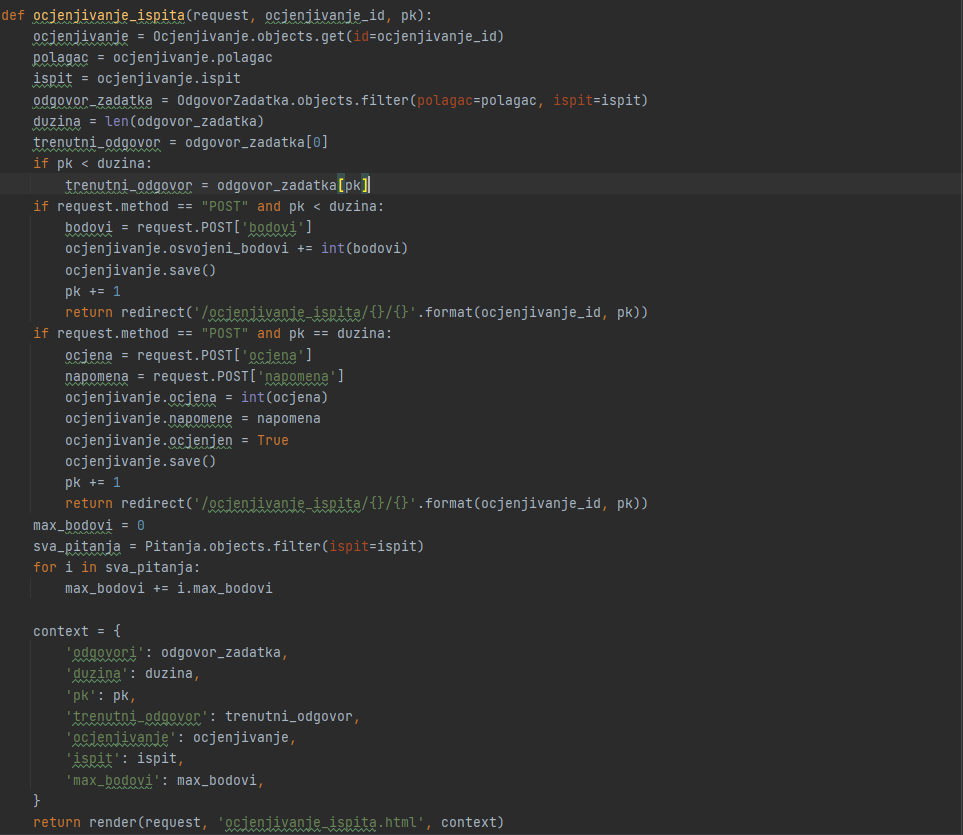
U slučaju odabira „Zadavač“, korisnik će imati 3 opcije na početnoj strani:

* Napravi ispit, koja mu omogućava da kreira ispit sa svim osnovnim podacima, kao i dodavanje pojedinačnih pitanja u taj ispit.
* Lista ispita, koja prikazuje sve ispite koje je taj korisnik napravio, i omogućava mu da pokrene bilo koji od njih
* Ocjeni ispit, prikazuje listu neocijenjenih ispita, i polagača tih ispita, i omogućava bodovanje i ocjenjivanje istih

Računi sa odabranom ulogom „Polagač“ imaju, na početnoj strani, samo opciju polaganja ispita.Ta opcija omogućava unos ispitnog koda, i zatim polaganja tog ispita odgovaranjem na pitanja onim redoslijedom kojim je to zdavač definisao. U slučaju unosa koda od ispita kojeg je taj korisnik već polagao, i ako je taj ispit ocijenjen, korisnik će onda vidjeti njegov rezultat, kao i bilo kakve napomene i kritike od strane zadavača.

## Backend

Pošto backend nije glavni fokus projekta, ovdje će samo biti ukratko opisana jedna funkcija iz views.py filea, s ciljem sticanja nekog osnovnog razumijevanja o razvoju backenda ove aplikacije. Funkcija ocjenjivanje\_ispita ima dosta elemenata koji se ponavljaju kroz cijeli projekat, kao i značajan broj upita za bazu podataka, pa će zbog toga ona ovdje biti prikazana.



SLIKA 6 – PRIMJER KODA ZA FUNKCIJU U APLIKACIJI

Dakle, kao što ime i govori, ova funkcija opisuje proces korisnikovog ocjenjivanja nekog polaganog ispita. U definiciji funkcije, pored standardnog parametra request koji nam je za svaku funkciju potreban, imamo i druga dva parametra. U suštini ovi parametri opisuju kako će URL izgledati prilikom poziva ove funkcije. Parametar pk nam označava koji se zadatak trenutno ocijenjuje. Prvo što radimo jeste slanje upita bazi da bismo dobili osnovne potrebne podatke, kao što je Ocjenjivanje objekat (ovi objekti se kreiraju dok polagač polaže ispit, da bismo mu mogli odmah bodovati pitanja tima zaokruži) i OdgovorZadatka objekat. Uzimamo sve objekte iz OdgovorZadatka koji imaju trenutnog polagača i ispit koji se ocjenjuju koristeći naredbu filter. Naredba filter će uvijek kreirati listu kada vrati rezultate upita, čak i ako ima samo jedan objekat kao rezultat upita. Sada prolazimo kroz zadatke, i uzimamo vrijednosti bodova koje je korisnik dodijelio sa frontenda, i spremamo te vrijednosti povećavanjem atributa osvojeni\_bodovi. Ako naš pk parametar dostigne vrijednost ukupnog broja zadataka, onda korisniku šaljemo formu u kojoj dodjeljuje ocjenu na osnovu dosad osvojenih bodova, kao i opciju da doda neku napomenu. Na kraju to sve spremimo u bazu podataka. Da bismo korisniku pokazali maksimalan broj bodova koji se mogu osvojiti na njegovom ispitu, moramo proći kroz sva pitanja u ispitu i sabrati njihove maksimalne bodove. Na kraju imamo dictionary (rječnik) varijablu u kojoj spremamo sve varijable koje želimo koristiti u frontendu, i dodjeljujemo im ključ kojim ćemo im pristupati. Tu varijablu onda šaljemo kao return cijele funkcije, zajedno sa HTML formom koja je vezana za ovaj URL.

Ovo je naravno bilo kraće i jednostavnije objašnjenje samo jedna od nekoliko funkcija koje čine ovaj file, ali trebalo bi biti dovoljno da se postigne neko osnovno razumijevanja rada u backendu ove aplikacije.

Pored views.py filea, važno je spomenuti i:

* urls.py, u kojem definišemo naše URL linkove, i povezujemo ih s funkcijama iz views.py
* admin.py, ovdje definišemo izgleda admin stranice na kojoj nadgledamo našu bazu, i možemo raditi stvaranje, update, čitanje i brisanje podataka iz baze
* forms.py, omogućava razvoj formi koje korisnik popunjava, obično se te forme povežu s nekim modelom, tako da odmah čitav objekat možemo napraviti i spremiti u bazu

# Zaključak

Dakle, cilj ovog projekta je bio razvoj i implementacija baze podataka za aplikaciju koja omogućava online održavanje ispita. Razvoj baze je urađen skoro potpuno, s tim da ako bi porasle potrebe aplikacije, bilo bi moguće i samu bazu proširiti. Puni potencijal ove relativno kompleksne baze nije u potpunosti iskorišten u aplikaciji, jer to nije bio fokus projekta. Ipak, većina podataka iz baze je dostupna u interfaceu aplikacije, a svako polje u bazi je barem u jednom od dijelova aplikacije iskorišteno za unos podataka. Proces normalizacije je detaljno opisan, ali početna verzija baze je zadovoljila prve tri normalne forme, pa u poglavlju o normalizaciji je samo objašnjeno zašto je svaka forma zadovoljena.

# Literatura

1. <https://docs.djangoproject.com/en/4.1/>
2. <https://www.python.org/>
3. <https://www.javatpoint.com/dbms-normalization>
4. <https://stackoverflow.com/>
5. <https://dbeaver.com/docs/wiki/>